

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using SoundPlayback2;
using SoundCapture;
using System.Threading;
using System.Windows.Threading;

namespace Gameplay
{
    //tconductor class and the scoreArgsClass used for the endOfSongEvent
    /*
    * The TConductor is the overall 'warapper class' or 'top level class' for the gamePlay element of the
    * game. TConductor controlls all aspects from calling scrolling the viewport image (of the dynamically
    * generated TAB, listening for the TGuitarFrequency event to compare what the user has played to the
    * current note in the song. TConductor also keeps track of the score and at the end of the song
    * fires an event that passes the scoring details and controll which in our case is a form.
    */
    public class TConductor
    {
        /*TConductors Main Components, the song information, the song image (dynamically generated)
        *the way in which the image is displayed, comparison tools*/
        public TSong mySong;
        public TScore myScoreImage;
        public TViewport myViewport;
        public List<TComparer> notesToCompare;
        private TGuitarFrequency GuitarAnalyser;
        public scoreCard currentScoreCard;

        //TConductors timer and variables
        DispatcherTimer myTimer = new DispatcherTimer();
        public double TimerCount;
        public const int baseTimerValueAt120 = 500;

        //viewport properties
        public Canvas theViewportArea;
        public int viewportOffSet;
        public const int viewportWidth = 1024;
        public Window theActualForm;

        //developers feedback tools
        public ListBox targetListBox;
        public ListBox targetListBox2;

        //public SoundPlayer Player;
        public TMusicPlayer Player;
        public String theWavFile;

        //real time feedback stuff and scoring
        TBaseNote currentSongNote;
        int currNoteLoc;
        public int currentNoteHits;
        public int currentNoteMisses;
        public int currentNoteMatches;
        public bool atLeastOnce;

        //Score feedback labels
        public Label scoreLabel;
        public Label multiplierLabel;

        //The feedback boxes to illustrate weather a user gets the note right or wrong
        public Canvas feedbackBox;
```

```

public Canvas feedbackBox2;
public Canvas feedbackBox3;

public SolidColorBrush whiteBrush;      /*making 4 types of brushed here so the overhead will*/
public SolidColorBrush redBrush;        /*be in the constructor so when the game is running */
public SolidColorBrush greenBrush;      /*all you have to do is assign the colour to the feedback */
public SolidColorBrush orangeBrush;     /*box as apposed to making a brush and assigning three colors to it each time */

//delegate method used to manage cross threading issues
public printer myPrinter;

//the users playback device
PlaybackDevice playDevice;

//the source location for the musicXML document
String theMusicXMLDoc;

//image source values- used for flaging the change to the next source image and specifying the current one
public int pageTurnFlag;
public int pageTurnIndex;

//*****

/* This is the TConductor constructor where all initial values are created and instantiated.
 * This is where the Conductor also creates all nessecary parts for the the user to play a
 * song. The song notes holder is created and populated, The TAB holder is created and
 * generates its image, the audio is loaded and buffered, the timer used for the song is created
 * and has an appropriate method added to its 'tick' event.*/
public TConductor(String theMusicXMLDoc, String theWavFile, Window theActualForm, ListBox targetListBox, ListBox targetListBox2, PlaybackDevice playDevice, Device capDevice, Canvas
theViewportArea, Canvas feedbackBox, Canvas feedbackBox2, Canvas feedbackBox3,Label scoreLabel, Label multiplierLabel)
{

    //This is setting up the playback device
    this.playDevice = playDevice;

    //scoring feedback labels set
    this.scoreLabel = scoreLabel;
    this.multiplierLabel = multiplierLabel;

    //The actual form where the canvas is held is noted- also used for navigation purposes
    this.theActualForm = theActualForm;

    //list boxes are assigned- used for developer testing
    this.targetListBox = targetListBox;
    this.targetListBox2 = targetListBox2;

    //viewport area is defined by the value passed in
    this.theViewportArea = theViewportArea;

    /* the TSong class is created. This is the class which holds all the note information
     * and is later drawn on to provide the data to generate the TAB and note comparisons*/
    this.theMusicXMLDoc = theMusicXMLDoc;    //setting the souce location of the MusicXML document
    this.theWavFile = theWavFile;           //setting the wav file location
    mySong = new TSong(theMusicXMLDoc);      //create the TSong class

    //Setting the note count values used for what position in the song it is up to
    //and also the default values that are used for real time note comparison (hit and miss count)
    this.currNoteLoc = 0;
    this.currentSongNote = mySong.songNotes[currNoteLoc];
    this.currentNoteHits = 0;
    this.currentNoteMisses = 0;
    this.currentNoteMatches = 0;

    //flag that signals that at least one note hit was correct, potentially 1 correct and 1 wrong = right. but 0 right and 0 wrong = wrong.
    this.atLeastOnce = false;

    //feedback box set up- this is what provides the real time feedback
    this.feedbackBox = feedbackBox;
    this.feedbackBox2 = feedbackBox2;
    this.feedbackBox3 = feedbackBox3;

    //pre setting the colours used to display in the boxes so the overhead is taken care of here
    //and not recreated each timer tick/note detection where feedback is given.
    this.whiteBrush = new SolidColorBrush(Colors.White);

```

```

        this.redBrush = new SolidColorBrush(Colors.Red);
        this.greenBrush = new SolidColorBrush(Colors.Green);
        this.orangeBrush = new SolidColorBrush(Colors.Orange);

        //The list of TComparer which is populated by the Frequency dected section of code
        notesToCompare = new List<TComparer>();

        //Creating the dynamically generated TAB
        myScoreImage = new TScore(mySong, theViewportArea);

        //This is currently not being used but a potential solution to the LAG issues
        this.viewportOffset = (8 * (mySong.smallestDivision * 4));

        //Creating the Viewport, what scrolls the images of TAB
        myViewport = new TViewPort(0, 0, viewportWidth, 350, myScoreImage.renderImages);

        //Create the timer
        myTimer = new DispatcherTimer();
        myTimer.Interval = TimeSpan.FromMilliseconds(Convert.ToDouble(baseTimerValueAt120 / mySong.smallestDivision));
        myTimer.Stop();
        this.myTimer.Tick += new System.EventHandler(this.myTimer_Tick);

        //Creating the sound player passig the form it is to be used on and the audio file it plays
        Player = new TMusicPlayer(playDevice, theWavFile, theActualForm);

        // Creating the sound capture class used to detect what the user plays
        GuitarAnalyser = new TGuitarFrequency(capDevice, this);

        //Set the timer count of the piece of music, this keeps track of where we are in the piece
        TimerCount = 0; //0 or 1??

        //Adding the prinout method to the myPrinter Delegate method to solve cross threading issues
        myPrinter = new printer(printout);

        //create the score card
        currentScoreCard = new scoreCard(mySong.numberOfNotes);
    }

    //*****

    /*This wrapper functino starts the song, starting the guitar capture, the timer of TConductor and starts the Aidio*/
    public void startTest()
    {
        //setting up the start values for the image structure (array of source images 10240px long)
        pageTurnIndex = 0;
        pageTurnFlag = myScoreImage.ScoreImageFlags[pageTurnIndex];

        //start the sound capture
        GuitarAnalyser.StartCapture();

        //repeat 3 times in an attempt to bring the audio into memory to avoid lag
        for (int i = 0; i < 3; i++)
        {
            Player.Play(); //play the audio
            while (Player.Player.BufferingProgress < 1) //Buffer the aidio untill it is completly buffered
            {
                //int i = 0;
            }
            Player.Stop(); //stop the audio
        }

        Player.Play(); //start the audio for real
        while (Player.Player.BufferingProgress < 1) //attain a buffer
        {
            //int i = 0;
        }

        myTimer.Start(); //this starts the song and all that goes with it ie viewport etc
    }

    //*****

    /*This function is called when there is a need to stop the guitar capture, audio and timer*/

```

```

public void stop()
{
    myTimer.Stop();           // stop the timer
    Player.Stop();           //stop the audio
    GuitarAnalyser.StopListenning(); //stop the capture
}

//*****

/*This function is Called from TGuitarFrequency.cs when a note has been detected.
 * a note name 'string' is passed in representing what the user just played.
 * This note is combined with the current time (TConductors 'TimerCount') and is sent off to the
 * updateScoreCard function which compares the note with the song note and updates the score card accordingly*/
public void buildUserNoteForCompare(String noteName)
{
    String userNote = noteName;
    double time = TimerCount;

    TComparer note = new TComparer(userNote, time);

    updateScoreCard(note); //call to the real time feedback version of comapre and give feedback
}
//*****

/* This is the functino that is assigned to the delgate method that is used to print the feedback
 * information to the screen while avoiding the cross threading errors- it recieves a string
 * that containt either the text 'Hit', 'Not correct' or 'Mid' each one fires a different action*/
public void printout(String output)
{
    targetListBox2.Items.Add(output);

    switch(output)
    {
        case "Hit":
            feedbackBox.Background = whiteBrush;
            feedbackBox2.Background = whiteBrush;
            feedbackBox3.Background = greenBrush;
            break;

        case "Not correct":
            feedbackBox.Background = redBrush;
            feedbackBox2.Background = whiteBrush;
            feedbackBox3.Background = whiteBrush;
            break;

        case "Mid":
            feedbackBox.Background = whiteBrush;
            feedbackBox2.Background = orangeBrush;
            feedbackBox3.Background = whiteBrush;
            break;
    }

    scoreLabel.Content = currentScoreCard.score;
    multiplierLabel.Content = currentScoreCard.multiplier;
}

//*****

//The delegate method must accept method that return void and accept a string- Used
//for the feedback method aboce
public delegate void printer(String output);

//*****

/*this i a function that returns true or false to weather or not the passed in TComparer
 * song note (class built around what note the user has just played) matches the current
 * note in the actual piece of music. This loops through all of the song details (possible notes)
 * to see if the user note matches any- if so it returns true (this is our redementary chord recognition
 * - Advanced chord recognition would see song note (list of note info) passed to the chord detection
 * specific function where the user sound data would be checked against the target chord frequency*/
public bool sameNoteComparison(TComparer userNote)
{
    bool match = false;

```

```

        //to make this faster could change it so that once found it stops looking any further
        foreach (TNoteDetail t in currentSongNote.myNoteDetails)    //loop through every note detail
        {
            //if any of the current note information matches then set the note comparison test to true
            if (userNote.theNote == t.theNote)
                match = true;
        }

        return match;
    }

//*****

/* the new version of 'compare and show func'
 * This function is called after the user note has been created after being called by TConductors TGuitarFrequency
 * after a frequency has been detected.
 * With the current user played note it is compared to the current note in the song and updates the score card accordingly*/
void updateScoreCard(TComparer userNote)
{
    //when trying to get the computer to listen to the audio it is playing this does not work on home computer.
    String debugging = "false";

    if (sameNoteComparison(userNote))    //if the currently played user note is the same as the song note
    {
        atLeastOnce = true;
        currentNoteMatches += 1;    //increment match hits (hits and misses effect this)
        currentNoteHits++;    //increment hit counter- used for debugging our hits and misses idea    //doesnt seem to be working
        debugging = "true";    //debugginh variable

        currentScoreCard.numberOfTotalHits++;

        currentScoreCard.score += (5 * currentScoreCard.multiplier);    //add to the current score
    }
    else
    {
        currentNoteMatches -= 1;    //if not the same decrement
        currentNoteMisses++;    //increment miss counter- used for debugging our hits and misses idea

        currentScoreCard.numberOfTotalMisses++;
    }

    //building an output screen to display feedback
    String current = "Song note: " + currentSongNote.ToString() + ". User note: " + userNote.theNote + ". Timer: " + TimerCount.ToString() + " " + debugging + " " + currentNoteMatches.
ToString();

    feedbackBox.Dispatcher.Invoke(myPrinter, current);

    //real time feedback is fired here: this calls the dispatcher for the feedback boxes etc
    //to illustrate to the user how they are doing on the current note
    realTimeFeedback();
}
//*****

/*This is the realTimeFeedback function:
 *After a note has been recognised it is compared and alters the score card
 *then this is the visual side of the note comparison where the current notes
 *accumulated number of hits and misses are computed to determine what feedback to give*/
public void realTimeFeedback()
{
    //if the current note matches is above the threshold value (1) then..
    if (currentNoteMatches >= 1)
    {
        //calling dispatcher so that cross threading problems dont occur and provide feedback for a 'hit'
        feedbackBox.Dispatcher.Invoke(myPrinter, "Hit");
    }
    else
    {
        //provide feedback for a 'mid' - this means in between hit and miss
        if ((currentNoteMatches >= -1))
        {
            feedbackBox.Dispatcher.Invoke(myPrinter, "Mid");
        }
        else    //otherwise is less than the lowest cut off so fail
        {

```

```

        feedbackBox.Dispatcher.Invoke(myPrinter, "Not correct");
    }
}

//*****

//This is the function that controlls the viewport stuff each timer tick
public void checkForTurnPage()
{
    if ((TimerCount + 1) >= (pageTurnFlag))
    {
        //move to next source, viewport method
        pageTurnIndex++;
        pageTurnFlag = myScoreImage.ScoreImageFlags[pageTurnIndex];
        myViewport.moveToNextSource();
    }
}

//*****

/*This is the timer tick of the TConductor. This method is fired on the TConductors Timers-tick.
This method is used to controll:
* the location of the image/TAB scrolling across the screen
* the current 'music note' in the song which is used for comparison purposes in the TConductor.TGuitarFrequency.FrequencyDetected => compare and show function*/
public void myTimer_Tick(object sender, EventArgs e)
{

    myViewport.ScrollTabImage();
    myViewport.ViewPortDraw(theViewportArea);

    TimerCount++;

    checkForTurnPage();

    //if end of current note then increment to the next note
    if ((currentSongNote.theTime + currentSongNote.theDuration) == (TimerCount))
    {
        currNoteLoc++; //incrementing the note location
        endOfNoteCheck();

        if (currNoteLoc >= mySong.songNotes.Length) //if the last note in the song-stop
        {
            currNoteLoc = 0; //reset current song note location

            currentScoreCard.checkStreak(); //need to recheck here incase you got all notes right otherwise you highest streak will still be 0!!

            feedbackBox.Dispatcher.Invoke(myPrinter, "end of song");

            endOfSong(); //call the function to deal with the end of the song and fire the forms event
        }
        else //else- note end of song so reset curr note hits and misses etc
        {
            currentScoreCard.checkStreak();
            resetNoteHitCounters();
        }
    }

    //back up end of song handler if note count is to fail
    if (TimerCount >= ((mySong.numberOfBars * (mySong.smallestDivision * 4) ) + 2) )
    {
        //max number of notes per bar (number of movements/scrolls per bar)
        endOfSong();
    }
}

//*****

//reset the note count values - called on the start/end of a new note
public void resetNoteHitCounters()
{
    currentSongNote = mySong.songNotes[currNoteLoc]; //set the 'next' note in the song to the 'current' note
    currentNoteHits = 0; //reset the current number of hits for the note
    currentNoteMisses = 0; //reset the current number of misses for the note
}

```

```

        currentNoteMatches = 0; //reset the current note matches.. (whats this one??) a value that is incremented AND decremented based on hits and misses as apposed to
Just counting the hits or missies
        atLeastOnce = false; //set the at least once flag back to false

    }

    //*****

    //Function to check if the current note in the song is at the end of its duration based of the
    //TConductors beatcount/ Timer. If it is the end update the scorecard acrodingly for a hit/missed note
    public void endOfNoteCheck()
    {
        //if the current note matches are above the threshold (0) and there has been a correct note recognised at least once (so the user cant play nothing and get it all right) then... update
score card for correct note
        if ((currentNoteMatches >= 0) && (atLeastOnce))
        {
            currentScoreCard.numberOfCorrectNotes += 1; //end of song note- check if they got enough 'hits' and update score card

            currentScoreCard.score += 50 * currentScoreCard.multiplier; //update score card score

            currentScoreCard.currentStreak += 1; //update streak
        }
        else
        {
            currentScoreCard.checkStreak();

            currentScoreCard.currentStreak = 0;
        }
    }

    //*****

    public delegate void endOfSongHandler(object sender, scoreEventArgs e);
    public event endOfSongHandler passToFormResults;

    //*****

    /*This function is fired when the end of a song is reached.
    * It notifies the programmer with a message box for now
    * Calls the 'stop' method to deal with stoping all the audio, capture and scrolling
    * it creates the scoreEventArgs which are used in firing the pass to form event which
    * invertibly passes controll to the form sending out the users score results
    */
    public void endOfSong() //function to deal with the end of the song
    {
        stop(); //stop listening displaying

        scoreEventArgs testArgs = new scoreEventArgs(true, currentScoreCard);

        if (passToFormResults != null) //if the event is empty/no methods that will fire from it
        {
            passToFormResults(this, testArgs);
        }
        else
        {
            MessageBox.Show("Nothing was assigned to the TConductor end of song event!!");
        }
    }

    //*****

    public class scoreEventArgs : EventArgs
    {
        public bool pass;
        public scoreCard playerScoreCard;

        public scoreEventArgs(bool pass, scoreCard playerScoreCard)
        {
            this.pass = pass;
            this.playerScoreCard = playerScoreCard;
        }
    }
}

```

